DOCKER, KONTEYNER TEKNOLOJİSİ NEDİR?

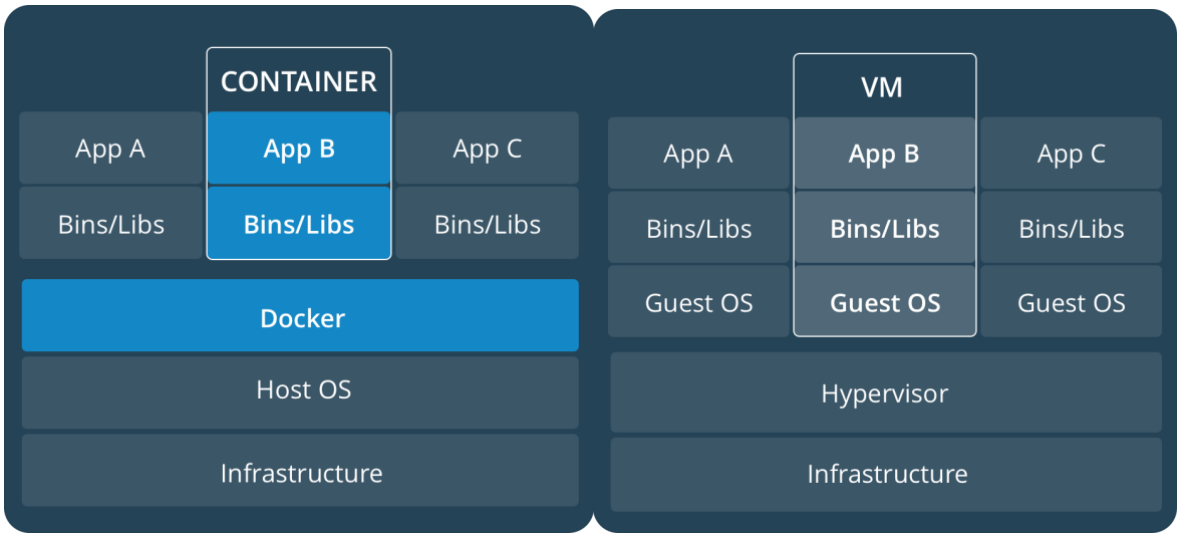
Yazar: Samet SAZAK
Baskı: 2018

İÇİNDEKİLER

Docker ve Konteyner teknolojisi nedir?

Docker, konteyner teknolojisi kullanarak uygulama oluşturma, dağıtma ve çalıştırma işlemlerini kolaylaştırmak için tasarlanmış bir araçtır. Konteynerler, bir geliştiricinin bir uygulamayı, kütüphaneler ve diğer bağımlılıklar gibi ihtiyaç duyduğu tüm parçalarla paketlemesini ve tek bir paket olarak göndermesini sağlar. Bu sayede uygulama içerisinde kod yazma ve test etmek için kullanılan makineden farklı olabilecek herhangi bir özelleştirilmiş ayardan bağımsız olarak, uygulamanın başka bir Linux makinesinde (konteynerde) çalışmasını sağlamaktadır.

Docker bir sanal makine gibidir ancak sanal bir makineden farklı olarak, bütün bir sanal işletim sistemi oluşturmak yerine, Docker uygulamaların aynı Linux çekirdeğini kullandıkları sistem olarak kullanmasına izin verir. Bu önemli bir performans artışı sağlar ve uygulamanın boyutunu azaltır.

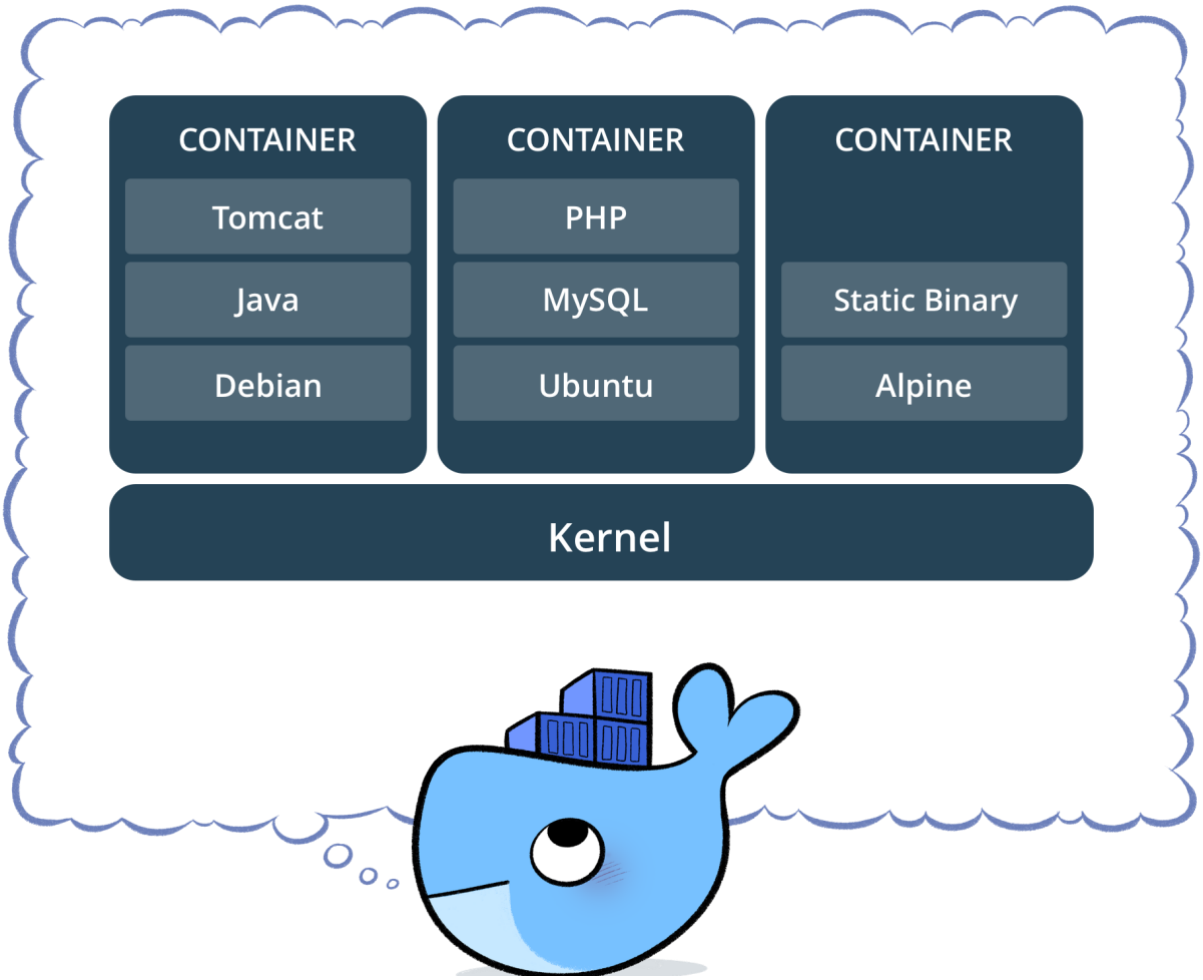


(https://docs.docker.com/get-started/#containers-and-virtual-machines)

Docker açık kaynak bir projedir. Bu da Herkesin Docker'a katkıda bulunabileceği ve kutunun dışında mevcut olmayan ek özelliklere ihtiyaç duyması halinde kendi ihtiyaçlarını karşılamak için genişletebileceği anlamına gelir.

Konteyner nedir?

Bir konteyner imajı, çalıştırmak için gereken her şeyi içeren bir yazılım parçasının hafif, bağımsız, yürütülebilir bir paketidir. Kod, çalışma zamanı, sistem araçları, sistem kütüphaneleri, konfigurasyonlar bulundurur. Hem Linux hem de Windows tabanlı uygulamalar için mevcut olan konteynerli yazılımlar, çevreye bakılmaksızın her zaman aynı şekilde çalışır.



(https://docs.docker.com/get-started/#containers-and-virtual-machines)

Konteynerler, çevresiyle ilgili yazılımı izole eder, örneğin geliştirme ve aşamalandırma ortamları arasındaki farklılıklar ve aynı altyapı üzerinde farklı yazılımlar kullanan ekipler arasındaki sorunları azaltmaya yardımcı olur.

Temel Docker Güvenliği

Siber Güvenlik, bugünün veri merkezlerinde ve iyi bir nedenden ötürü en önemli konumdadır. Oldukça küçük ölçekli bir ihlal bile çok büyük hasarlara yol açabilir. Bu nedenle, kuruluşların özellikle güvenlik bulgularına ve yeni güvenlik açıklarına karşı duyarlı olmaları gerekmektedir. Sanallaştırma ana akım haline geldiğinde, VM güvenliği ortak bir konuydu. BT Güvenlik uzmanları, sanallaştırılmış bir ortamın potansiyel zayıflıklarını uzun bir süredir tartışmaktadır.

Belki de en kötü senaryo “VM kaçışı (VM Escape)” dır. Bu terim, bir saldırganın bir guest sanal makinesini tehlikeye attığı ve sanal makinenin içerisinden “kaçılabildiği” ve hipervizör üzerindeki diğer ana işlemlere erişebileceği bir durumu tanımlamak için kullanılır.

Sanallaştırma liderleri, bu tür güvenlik sorunlarının ele alındığından emin olmak için büyük çaba sarf ettiler ve VM kaçışını gerçek bir tehdit olarak yüksek sesle reddettiler. Ve biz hepimiz onlara inanmaya başladığımızda, CrowdStrike'daki araştırmacılar, CEN-2015-3456 olarak adlandırdığımız ve “VENOM” olarak adlandırılan zafiyeti gösterdi ve QEMU'nun sanal disket sürücüsündeki bazı savunmasız kodları kötüye kullanarak, saldırganlar potansiyel olarak kazanma potansiyeline sahip oldular ve ana sistemde ve ana bilgisayarda çalışan diğer sanal makinelere erişim sağlayabildiler. Bu sebeple Docker bir sanallaştırma servisi olmasa da konteyner güvenliğini sağlama konusunda bu yazımızda yardımcı olmaya çalışacağız.

Zararlı İmajlardan Korunma Amaçlı Önlemler

İmzalanmamış İmajların Kullanılmaması

Docker ile, açık depolardan konteyner imajları indirebilirsiniz. Bu, konteynerleri kimin oluşturduğuna güvenmeniz anlamına gelmektedir. Konteynerin güvenli bir şekilde oluşturulduğunu nereden biliyorsunuz? Daha da kötüsü, konteynerin zararlı veya bozuk dosyaları içermediğini nereden biliyorsunuz? Bilmiyorsunuz. Bu nedenle Docker Hub ücretli planını kullanmayı düşünebilirsiniz. Bu ücretli hizmet, kullandığınız depoların tarandığından emin olmanın bir yoludur. Buna alternatif olarak indirdiğiniz konteynerlerin neleri yüklediği, üzerinde hangi servislerin çalıştığını kontrol etmek gerekmektedir.

Örneğin, bir Redmine imajı indirmek istediğimizi varsayalım, Docker hub içerisinde bulunan imajları aşağıdaki gibi aradığımızda birçok seçeneğin bizi beklediği görülmektedir. Fakat bunlar kullanıcılar tarafından oluşturulmuş imajlardır. Bu imajlara ilk etapta güvenmemek gerekmektedir.

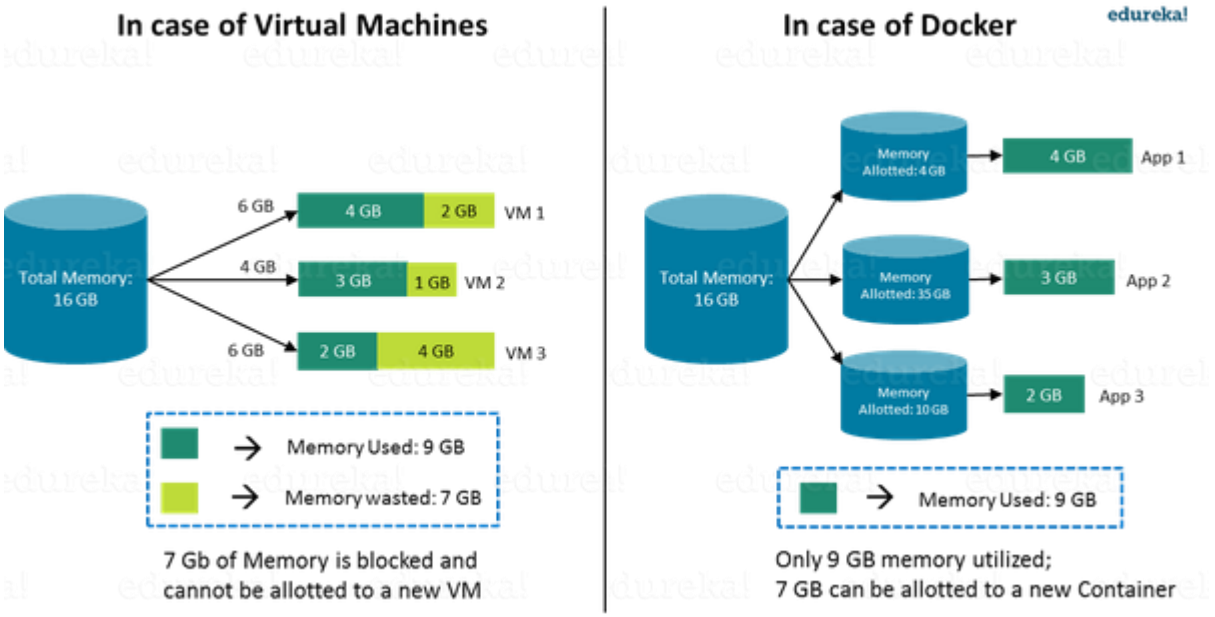
```
~ » docker search redmine                                                                    sametsazak@MacBook-Pro
NAME                          DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
redmine                       Redmine is a flexible project management web…   541       [OK]
sameersbn/redmine                                                             269                  [OK]
bitnami/redmine               Bitnami Docker Image for Redmine                26                   [OK]
74th/redmine-all-in-one       Redmine includes hosting SVN & Git , backlog…   9                    [OK]
inspiredgeek/redmine-alpine   Simple Docker images to run Redmine tracker …   4                    [OK]
turnkeylinux/redmine-13.0     TurnKey Redmine - Integrated SCM & Project M…   2
eeacms/redmine                EEA Redmine docker setup                        2                    [OK]
tkeydll/docker-redmine-backlogs Redmine with backlogs plugin.                 1                    [OK]
themill/redmine               fork for redmine to add/test theming            1                    [OK]
fjudith/redmine               Dockerized Redmine based on redmine:3.3 offi…   1                    [OK]
minimaru/openshift-redmine    A Debian8 based Redmine v3.2 image for use w…   1
commonms/redmine              Docker image for Redmine.                       1                    [OK]
starfox/redmine-plugin-dashboard A container designed to install redmine-dash… 1                    [OK]
trollin/redmine                                                               0
rubykube/redmine              Redmine 4.0 container                           0                    [OK]
honsiorovskyi/redmine         Official Redmine + Git + Mercurial              0                    [OK]
stackbrew/redmine             Deprecated; use 'redmine' from https://hub.d…   0
thiagorider/redmine           Redmine Docker Image Automated Build Repo.      0                    [OK]
nitra/redmine_priority_tasks  redmine_priority_tasks                          0                    [OK]
sorintdev/redmine             Redmine with custom theme                       0
togent2/redmine_ttdlx_enabled redmine_ttdlx_enabled                           0                    [OK]
robobeerun/redmine            Redmine image that runs as non-privileged us…   0                    [OK]
mikroways/redmine             redmine passenger image                         0                    [OK]
amd64/redmine                 Redmine is a flexible project management web…   0
i386/redmine                  Redmine is a flexible project management web…   0
~ »                                                                                          sametsazak@MacBook-Pro
```

Docker 1.8'den itibaren "Docker Content Trust" adı verilen yeni bir güvenlik özelliği getirildi. Bu özellik "Docker Hub" deposunda bulunan tüm Docker imajlarının orijinalliğini, bütünlüğünü ve yayınlanma tarihini doğrulamanıza olanak tanımaktadır Bu içerik güvencesi varsayılan olarak etkin değildir. Etkinleştirildiğinde Docker, imzalanmamış imajların hiçbirini indirememektedir.

Bu özelliği etkinleştirmek için, sudo export DOCKER_CONTENT_TRUST = 1 komutunu çalıştırabilirsiniz. Artık imzalı olmayan bir imajı indirmeye çalıştığınızda Docker sizi bilgilendirecektir.

DOS (Denial of Service) Saldırılarından Korunma Amaçlı Önlemler

Konteynerler İçin Kaynak Sınırlaması



(https://www.docker.com/what-container)

Varsayılan olarak, bir konteynerde kaynak kısıtlaması yoktur ve ana bilgisayar izin verdiği ölçüde belirli bir kaynağın çoğunu kullanabilir. Docker, konteyner çalıştırma komutu konteynerin kullanabileceği bellek, CPU veya blok miktarını kontrol etmenin yollarını sunar. Bir konteyner çalışmasında sorun oluştuğunda ve tüm hostun kaynaklarını tüketmeye başladığı bir senaryo üzerinde kötü sonuçlar doğması muhtemeldir. Konteynerleriniz için kaynak limitlerini "docker run" komutundan ayarlanabilmektedir.

Memory ve CPU

Örneğin, bir konteyneri 1 GB bellek ile limitlemek istiyorsanız çalıştırma komutuna —memory = "1000M" seçeneğini ekleyebilirsiniz. Ayrıca CPU sayısını —cpus = X (X'in konteyner için kullanabileceğiniz CPU sayısıdır) ekleyerek sınırlayabilirsiniz.

CPU Cycle

Varsayılan olarak, tüm konteynerler eşit oranda CPU (cyle)döngüsü elde etmektedir. Bu oran değiştirilebilir. Konteynerin CPU paylaşım ağırlığı, diğer tüm çalışanların ağırlığına göre ayarlanabilmektedir.

Konteynerler varsayılan olarak 1024 gibi bir değerde tanımlanır. -c parametresi bu amaçla kullanılmaktadır.

```
$ docker run -d -c 512 myimage
```

CPU, Memory ve Realtime Scheduler konfigurasyonlarını detayları için bakınız: (https://docs.docker.com/config/containers/resource_constraints/)

Yetki Yükseltme ve Kernel Exploiting'e Karşı Önlemler

Konteynerler Arası Gereksiz İletişimi Engelleme

Eğer uygulama konteynerleri dış dünyayla veya diğer konteynerlerle konuşmaya ihtiyaç duymuyorsa, bunları nispeten güvenli ve izole bir ortamda çalıştırmak önemlidir. Bir konteyner saldırıya uğramış veya ele geçirilmiş ise izole edilmiş bir sistemde diğer konteynerlere veya uygulamaları etkileyemez veya soruna neden olmaz.

```
$ docker -d --icc=false --iptables
```

Komutu ile bu önlem alınabilmektedir. Daha fazla bilgi için (https://docs.docker.com/v17.09/engine/userguide/networking/default_network/container-communication/#communication-between-containers)

Root Kullanıcısını Kullanmamak ve Konteynere Kullanıcı Eklemek

Bir konteynerde çalışan bir işlemin, Linux'ta çalışan diğer işlemlerden farklı olmadığı unutulmamalıdır; iki sistem arasında farklı olarak konteyner sistem olduğunu bildiren küçük bir meta veri parçası vardır. Bu nedenle, bir konteynerde çalışan herhangi bir şey "host" sunucu üzerinde çalışan herhangi bir şeyle aynı şekilde ele alınmalıdır.

Sunucunuzda root olarak hiçbir şeyi çalıştırmamanız (ya da çalıştırmamalısınız) gibi, sunucunuzdaki bir konteynerde de "root" olarak hiçbir şey çalıştırmamalısınız. Başka bir yerde oluşturulan binary dosyaları çalıştırmak güven gerektirir ve bu konteynerdeki dosyalar için de geçerlidir. "Root" kullanıcı ile işlem yapmak yerine, Dockerfile'ınızda bilinen bir "UID" ve "GID" ile bir kullanıcı oluşturmak önemlidir ve işlemlerin bu kullanıcı ile yapılması önerilmektedir. Bu şekilde oluşturulan imajların, kaynaklara erişimi sınırlandırarak güvenli bir şekilde çalışması daha güvenlidir.

Örnek bir kullanıcı oluşturmak için:

```
FROM debian:stretch
RUN groupadd -g 999 testgroup && \
   useradd -r -u 999 -g testgroup testuser
USER testuser
CMD ["cat", "/tmp/gizli_dosya.txt"]
```

Linux Kabiliyetlerini Limitlemek

Docker run - cap-drop seçeneğini kullanarak, konteyner içinde sınırlı erişime sahip olması için bazı Linux kabiliyetlerini düşürerek konteyneri kendi içerisine kilitleyebilirsiniz.

Linux kabiliyetleri nelerdir?

Kabiliyetler (capabilities) man sayfasına göre bağımsız olarak etkinleştirilebilen veya devre dışı bırakılabilen farklı ayrıcalık birimleridir. Bir docker kapsayıcısında ayrıcalıklı süreçlerin kullanabileceği varsayılan kabiliyetler listesi:

chown, dac_override, fowner, fsetid, kill, setgid, setuid, setpcap, net_bind_service, net_raw, sys_chroot, mknod, audit_write, setfcap gibi birimlerdir.

İhtiyacınız olmayan yetenekleri kaldırmak yerine, ihtiyaç duyduğunuz yetenekleri ekleme yaklaşımını daha önemli olmaktadır.

Bu kabiliyetler hakkında daha detaylı bilgi için: (http://man7.org/linux/man-pages/man7/capabilities.7.html)

Bu yetenekleri docker kullanarak nasıl düşürebiliriz? Öncelikle, bir process'in sahip olduğu kabiliyetleri görelim. Linux'ta, Fedora üzerinde "libcap-ng-utils" paketinde bulunan bir process'in pscap olarak adlandırdığı kabiliyetleri görmenmize yardımcı olabilecek harika bir araç bulunmaktadır.

pscap kullanarak örnek çıktı pscap | head -10:

```
ppid  pid   name       command        capabilities
1   1082  root     systemd-journal   chown, dac_override, dac_read_search, fowner, setgid,
setuid, sys_ptrace, sys_admin, audit_control, mac_override, syslog, audit_read
1   1116  root     systemd-udevd   full
1   1760  root     auditd        full
1760  1778  root       audispd        full
1   1812  root     mcelog        full
1   1815  root     bluetoothd     net_bind_service, net_admin
1   1816  root     ModemManager    full
1   1817  root     systemd-logind  chown, dac_override, dac_read_search, fowner, kill,
sys_admin, sys_tty_config, audit_control, mac_admin
1   1818  root     rngd          full
```

Bir Docker Konteyner process'inin sahip olduğu kabiliyetleri aşağıdaki gibi görebilmekteyiz,

```
$docker run -d fedora sleep 5 >/dev/null; pscap | grep sleep
```

Çıktı sonucu:

```
26358 26375 root      sleep      chown, dac_override, fowner, fsetid, kill, setgid, setuid,
setpcap, net_bind_service, net_raw, sys_chroot, mknod, audit_write, setfcap
```

Eğer biz setfcap, audit_write, and mknod kabiliyetlerini düşürmek istersek konteyneri başlatırken --cap-drop=setfcap --cap-drop=audit_write --cap-drop=mknod parametrelerini kullanmalıyız.

```
$docker run -d --cap-drop=setfcap --cap-drop=audit_write --cap-drop=mknod fedora sleep 5
> /dev/null; pscap | grep sleep
```

Çıktı Sonucu:

```
26555 26571 root      sleep      chown, dac_override, fowner, fsetid, kill, setgid, setuid,
setpcap, net_bind_service, net_raw, sys_chroot
```

Bu kabiliyetlerin düşürülmüş olduğunu görüyoruz.

SELinux Kullanımı Ve Önemi

https://rhelblog.redhat.com/2017/01/13/selinux-mitigates-container-vulnerability/

Dosya Sistemini “Read-only (Salt-Okunur)” Moda Getirmek

Kapsayıcınızdaki dosyaları değiştirmeniz gerekmedikçe, dosya sistemini salt okunur hale getirebilirsiniz. Saldırıya uğrayan bir uygulama konteynerine sahip olduğunuzu düşünün. Saldırganın yapmak istediği ilk şey, istismar kodunun uygulamaya yazılmasıdır, böylece uygulama bir sonraki başlatılışında, istismar ile yerinde başlar. Eğer konteyner imajı salt okunursa, bir hacker, bir arka kapıyı yerinde bırakmaktan kaçınacak ve döngüyü en baştan başlatmak zorunda kalacaktır.

Docker, bunu bir süre önce "docker run -d --read only image" ile halletmek için bir özellik ekledi. Ancak kullanımı zor çünkü uygulamaların "/run" veya "/tmp" gibi geçici dizinlere yazılması gerektiğinde, bunlar salt okunur olduğunda uygulamalar başarısız olmaktadır. Buna çözüm olarak bu dizinleri tmpfs olarak bağlamak uygulamalarınızı çalıştırmanızı sağlayacaktır. Docker 1.10’dan itibaren şu şekilde bir açıklama ile sürülmüştür.

"Geçici dosya sistemleri: Artık --tmpfs parametresi kullanılıarak docker konteynerlerinde geçici dosya sistemleri oluşturmak gerçekten çok kolay. Bu, özellikle konteyner içindeki yazılım parçasının, bir salt okunur dosya sistemi ile çalışmasını beklerken yararlıdır. "

```
$docker run -d --read-only --tmpfs /run --tmpfs /tmp IMAGE
```

Docker'ın Çalıştığı Host Sunucu Üzerinde Yapılması Gereken Sıkılaştırmalar

Docker'ı güncellemek

Docker güncellemelerini güncel tutarak, Docker yazılımındaki güvenlik açıkları kapatılabilir. Bir saldırgan erişim elde etmeye veya yetki yükseltmeye çalışırken bilinen güvenlik açıklarından yararlanabilir. Düzenli Docker güncellemelerini yüklememek sizi savunmasız Docker yazılımı ile çalıştırmanıza neden olabilir. Yetki yükseltmeye, yetkisiz erişime veya diğer güvenlik ihlallerine yol açabilir. Yeni sürümleri takip edin ve gerektiğinde güncelleyin.

Önlem:
Aşağıdaki komutu çalıştırarak ve Docker sürümünün güncel olduğundan emin olun.

```
$ docker version
```

Referanslar

1. https://docs.docker.com/engine/installation/
2. https://github.com/moby/moby/releases/latest
3. https://github.com/docker/docker-ce/releases/latest

Docker processine sadece yetkili kullanıcıların erişmesi

Docker, konteynerlerin erişim haklarını sınırlamada ana bilgisayar (host) ve konteyner arasında dizin paylaşmanıza olanak tanır. Bu bir konteyneri /(kök) dizine bağlayarak başlatabileceğiniz anlamına gelmektedir. Konteyner daha sonra ana dosya sisteminizi herhangi bir kısıtlama olmaksızın değiştirebilir. Basit bir ifadeyle, sadece docker grubunun bir üyesi ayrıcalıklara haklara sahip olabildiği gibi daha sonra ana makinede (host) üzerinde kök dizini bir konteynere bağlayabileceği anlamına gelmektedir. Bu durumu kısıtlamak gerekmektedir.

Önlem: Docker ana bilgisayarında aşağıdaki komutu çalıştırın ve yalnızca güvenilen kullanıcıların docker grubunun üyeleri olduğundan emin olun.

```
$ getent group docker
```

1. https://docs.docker.com/engine/security/security/#docker-daemon-attack- surface
2. https://www.andreas-jung.com/contents/on-docker-security-docker-group- considered-harmful
3. http://www.projectatomic.io/blog/2015/08/why-we-dont-let-non-root-users-run- docker-in-centos-fedora-or-rhel/

Docker için Auditing Konfigurasyonunun Yapılması

Düzenli Linux dosya sisteminizi ve sistem çağrılarınızı denetlemenin (Auditing) yanı sıra Docker daemonunu denetlemek önemlidir. Docker daemon, root ayrıcalıklarıyla çalışır. Bu nedenle faaliyetlerini ve kullanımını denetlemek gereklidir.

Önlem: Docker daemon için bir Auditing kuralı olduğunu doğrulayın. Örneğin, aşağıdaki komutu çalıştırın:

```
$ auditctl -l | grep /usr/bin/docker
```

Bu kural size Docker ile alakalı bir auditing kuralı olup/olmadığını listeleyecektir.

Docker daemon'u için kural yoksa; /etc/audit/audit.rules dosyası içerisine:

```
-w /usr/bin/docker -k docker
```

Yazdıktan sonra, Auditd servisini yeniden başlatabilirsiniz.

Auditing oldukça büyük log dosyaları oluşturur. Periyodik olarak döndürüp arşivlediğinden emin olun. Ayrıca, kök dosya sistemini doldurmamak için ayrı bir auditing bölümü oluşturduğunuza dikkat edin.

Docker İçin Oluşturulması Gereken Auditing Kuralları:

```
-w /usr/bin/docker -k docker
-w /var/lib/docker -k docker
-w /etc/docker -k docker
-w /usr/lib/systemd/system/docker.service -k docker
-w /usr/lib/systemd/system/docker.socket -k docker
-w /etc/default/docker -k docker
-w /etc/docker/daemon.json -k docker
-w /usr/bin/docker-containerd -k docker
-w /usr/bin/docker-runc -k docker
```

Docker Daemon Sıkılaştırmaları

Konteynerler Arası Gereksiz İletişimi Engelleme

Varsayılan olarak, varsayılan ağ köprüsünde (network bridge) aynı ana bilgisayardaki (host) tüm konteynerler arasında sınırsız ağ trafiğini kullanabilmektedir. Böylelikle, her bir konteyner, aynı ana bilgisayar üzerindeki konteyner ağı boyunca tüm paketleri okuma potansiyeline sahiptir. Bu istenmeyen bir bilgi ifşasına yol açabilir. Bu nedenle, konteyner iletişimini network bridge üzerinde kısıtlayın.

Önlem: Aşağıdaki komutu çalıştırın ve varsayılan ağ köprüsünün (network bridge) konteyner içi iletişimi kısıtlayacak şekilde yapılandırıldığını doğrulayın.

```
$ docker network ls --quiet | xargs docker network inspect --format '{{ .Name }}: {{ .Options }}'
```

“Com.docker.network.bridge.enable_icc:false” şeklinde varsayılan olarak dönüş yapacaktır.

Bu durumu kısıtlamak için:

```
$ dockerd --icc=false
```

Kullanabilirsiniz, alternatif olarak Docker dokümanlarını takip edip özel bir ağ oluşturabilir ve yalnızca bu özel ağa iletişim kurması gereken konteynerleri birleştirebilirsiniz. --icc parametresi yalnızca varsayılan docker köprüsü için geçerlidir.

1. https://docs.docker.com/engine/userguide/networking/
2.https://docs.docker.com/engine/userguide/networking/default_network/container-communication/#communication-between-containers

Loglama Seviyesinin “Info” Olarak Seçilmesi

Uygun bir log seviyesi ayarlamak, daha sonra gözden geçirmek isteyeceğiniz olayları kaydetmek için Docker daemonunu yapılandırır. Bir temel logo seviyesi ve üstü, hata ayıklama (debug) logları dışındaki tüm logları yakalayacaktır. Gerekli olana kadar, Docker daemon'unu debug seviyesi loglama ile çalıştırmamalısınız.

Önlem: Docker’ı info seviyesi loglama ile çalıştırdığınızdan emin olun.

```
$ dockerd --log-level="info"
```

1. https://docs.docker.com/edge/engine/reference/commandline/dockerd/

Docker Servisinin Iptables Kurallarını Otomatik Olarak Düzenlemesi

Iptables, Linux çekirdeğindeki IP paket filtre kurallarının tablolarını oluşturmak, sürdürmek ve denetlemek için kullanılır. Docker sunucusu, gerekli izinleri otomatik olarak konteynerler için (ağ seçeneklerinizi nasıl seçeceğinize bağlı olarak) iptables üzerinde gereken değişiklikleri yapar. Konteynerler ve dış dünya arasındaki iletişimi engelleyebilecek ağ yapılandırmasını önlemek için Docker sunucusunun iptables ayarlarında otomatik olarak değişiklik yapmasına izin verilmesi önerilir. Ek olarak bu ayar Iptables'ı her seferinde güncellemenin zorluklarından da kurtarır.

Önlem: Docker tanımlı olarak dockerd --iptables=true olarak başlamaktadır. Kontrol olarak aşağıdaki komutu kullanabilirsiniz.

```
$ ps -ef | grep dockerd
```

1.https://docs.docker.com/engine/userguide/networking/default_network/container-communication/
2. https://fralef.me/docker-and-iptables.html

Güvenilmeyen Registry (kayıtların) Devre Dışı Bırakılması

Docker, özel bir kayıt defterini güvenli veya güvensiz olarak kabul etmektedir. Varsayılan olarak, kayıtlar güvenli kabul edilir. Güvenli bir kayıt birimi (registry) TLS kullanır. Kayıt defterinin CA sertifikasının bir kopyası Docket ana bilgisayarında (host) /etc/docker/certs.d/<registry-name>/ dizininde yerleştirilir.

Güvenli olmayan bir kayıt defteri (registry) geçerli kayıt sertifikasına sahip olmayan veya TLS kullanmayan bir kayıt defteridir. Production ortamında herhangi bir güvensiz kayıt (registry) kullanmamalısınız. Güvenli olmayan kayıtlar, production sisteminize olası bir bağlantı sağlanmasına sebep olabilir. Ayrıca, bir kayıt defteri güvensiz olarak işaretlendiyse, docker pull, docker push ve docker search komutları bir hata iletisine neden olmaz ve kullanıcı, potansiyel tehlike konusunda herhangi bir bildirim yapılmadan, bu kayıtlarla birlikte farketmeden süresiz olarak çalışabilir.

Önlem:

```
$ps -ef | grep dockerd
```

Docker'ın --insecure-registry parametresi ile başlatılmadığından emin olun. Örneğin, docker'ı şu şekilde başlatmamalısınız:

```
$dockerd --insecure-registry 10.1.0.0/16
```

1. https://docs.docker.com/registry/insecure/

Aufs Dosya Sisteminin Devre Dışı Bırakılması

Aufs, dosyalama sistemi en eski dosya sistemi sürücüsüdür. Aufs sürücüsü (driver) de bazı ciddi kernel çökmelerine neden olduğu bilinmektedir. En önemlisi, aufs, en son Linux çekirdeğini kullanan birçok Linux dağıtımında desteklenen bir sürücü değildir.

Önlem:

```
$docker info | grep -e "^Storage Driver:\s*aufs\s*$"
```

Örneğin Docker'ın "dockerd --storage-driver aufs" ile başlatılmaması gerekmektedir.

1. https://docs.docker.com/engine/userguide/storagedriver/selectadriver/#support ed-backing-filesystems
2. http://muehe.org/posts/switching-docker-from-aufs-to-devicemapper/
3. http://jpetazzo.github.io/assets/2015-03-05-deep-dive-into-docker-storage-drivers.html#1
4. https://docs.docker.com/engine/userguide/storagedriver/

Docker için TLS Authentication Konfigurasyonunun Yapılması

Docker daemonunu default (tanımlı) Unix socket dışında spesifik bir IP ve Port'u dinlenmesi sağlanabilmektedir. Docker sunucusuna IP ve port aracılığıyla erişimi kısıtlamak için TLS auhtentication (kimlik doğrulamasını) yapılandırılabilmektedir. Varsayılan olarak, Docker daemon ağa bağlı olmayan bir Unix soketine bağlanır ve kök (root) ayrıcalıklarıyla çalışır. Varsayılan docker daemon'unu bir TCP portuna ya da başka bir Unix soketine bağlarsanız, bu port ya da sokete erişimi olan herkes Docker daemon'una ve daha sonra ana sisteme tam erişime sahip olabilir. Bu nedenle Docker daemonunu başka bir IP / port veya Unix soketine bağlamamalısınız.

Docker daemon'unu bir ağ soketi üzerinden bağlamanız gerekiyorsa, Docker daemonunu Docker Swarm API'leri (kullanılıyorsa) için TLS kimlik doğrulamasını (auhtentication) yapılandırın. Bu, Docker daemon'unuzun ağ üzerindeki bağlantılarını TLS üzerinden başarıyla kimlik doğrulaması yapabilecek sınırlı sayıda istemciyle sınırlayacaktır.

```
$ps -ef | grep dockerd
```

```
--tlsverify
--tlscacert
--tlscert
--tlskey
```

Parametrelerinin aktif olduğundan emin olun. Varsayılan olarak TLS kimlik doğrulaması yapılandırılmamıştır.

1. https://docs.docker.com/engine/security/https/

Tanımlı Olan Ulimit Değerlerinin Düzenlenmesi

ulimit, Shell için mevcut olan kaynaklar ve onun tarafından başlatılan processler üzerinde kontrol sağlar. Sistem kaynak limitlerini ayarlamak sizi fork bomb gibi birçok felaketten kurtarır. Bazen, normal kullanıcılar ve bilinen processler bile sistem kaynaklarını aşırı kullanabilir ve sistemi kullanılamaz hale getirebilir.

Docker daemon için varsayılan ulimit ayarı tüm konteyner için tek bir ulimit konfigurasyonu kullanır. Her bir konteyner için ulimit kurmanıza gerek kalmaz. Ancak, gerekliyse, varsayılan ulimit konfigurasyonu konteyner çalışırken düzenlenebilir. Bu nedenle, sistem kaynaklarını kontrol etmek için, ortamınızda gerektiği gibi varsayılan bir ulimit tanımlayın. Eğer ulimitler düzgün ayarlanmamışsa, istenen kaynak kontrolü sağlanamayabilir ve hatta sistemi kullanılamaz hale getirebilir.

Örneğin aşağıdaki gibi bir tanım yapılabilmektedir:

```
$ dockerd --default-ulimit nproc=1024:2048 --default-ulimit nofile=100:200
```

Kullanıcı Namespace'lerini Aktifleştirmek

Docker daemon'daki Linux çekirdeği user namespace desteği, Docker ana bilgisayar (host) sistemi için ek güvenlik sağlar. Bir konteynerin, ana sistem tarafından kullanılan geleneksel kullanıcı ve grup aralığının dışında olan bir dizi kullanıcı ve grup kimliğine sahip olmasını sağlar.

Örneğin, kök kullanıcı konteynerin içinde beklenen yönetim ayrıcalığına sahip olacak, ancak ana bilgisayar sistemindeki ayrıcalıklı olmayan bir UID ile etkili bir şekilde eşleştirilebilecektir.

```
$ ps -p $ (docker inspect --format='{{ .State.Pid }}' <CONTAINER ID>) -o pid,user
```

Yukarıdaki komut, konteynerin PID'sini bulur ve daha sonra konteyner işlemiyle ilişkili host kullanıcıları listeler. Konteyner kök olarak çalışıyorsa, bu öneri uyumlu değildir.

Alternatif olarak, kullanıcıların Güvenlik Seçenekleri (Security Options) altında listelendiğinden emin olmak için docker info çalıştırabilirsiniz:

```
$docker info --format '{{ .SecurityOptions }}'
```

Tanımlı Cgroup Kullanımının Onaylanması

Sistem yöneticileri, genellikle, konteynerlerin çalışacağı varsayılan gruplarını tanımlar. Gruplar sistem yöneticileri tarafından açıkça tanımlanmasa bile, kapsayıcılar varsayılan olarak docker cgroup altında çalışır. Varsayılan olandan farklı bir cgroup eklemek mümkündür. Bu kullanım izlenmeli ve onaylanmalıdır. Varsayılan olandan farklı bir c grubu ekleyerek, kaynakları eşit olmayan şekilde paylaşarak ana bilgisayar (host) kaynak problemlerine neden olabilmektedir.

```
$ps -ef | grep dockerd
```

--cgroup-parent parametresinin aktif olmadığından emin olun. Varsayılan ayar yeterince iyidir ve olduğu gibi bırakılabilir. Özellikle varsayılan olmayan bir grup oluşturmak isterseniz, başlatırken docker daemonuna --cgroup-parent parametresi ile çalıştırın.

```
$ dockerd --cgroup-parent = / foobar
```

Docker Client Komutları İçin Yetkilendirmenin Etkinleştirilmesi

Docker'ın kullanıma hazır yetkilendirme modeli tam olarak yeterli olmamaktadır. Docker daemonuna erişim izni olan herhangi bir kullanıcı herhangi bir Docker client komutunu çalıştırabilir. Aynısı Docker'ın uzak API'sini kullanarak daemon ile iletişime geçmek isteyenler için de geçerlidir. Daha fazla erişim kontrolü gerekiyorsa, yetkilendirme eklentileri oluşturabilir ve Docker sunucu yapılandırmanıza ekleyebilirsiniz. Bir yetkilendirme eklentisi (plugin) kullanarak, Docker yöneticisi Docker uygulamasına erişimi yönetmek için ayrıntılı erişim ilkeleri yapılandırabilir.

1. Adım: Bir yetkilendirme eklentisi kurun / oluşturun.
2. Adım: Yetkilendirme politikasını istediğiniz gibi yapılandırın.
3. Adım: docker daemon programını aşağıdaki gibi başlatın:

```
$ dockerd --authorization-plugin = <PLUGIN_ID>
```

Her bir docker komutu, özel olarak yetkilendirme eklenti mekanizmasından geçer. Bu, küçük bir performans düşüşü getirebilir.

1. https://docs.docker.com/engine/reference/commandline/dockerd/#access- authorization
2. https://docs.docker.com/engine/extend/plugins_authorization/
3. https://github.com/twistlock/authz

Merkezi Ve Uzak Loglama Yapısının Yapılandırılması

Docker şimdi çeşitli loglama sürücülerini desteklemektedir. Logları depolamanın tercih edilen bir yolu, merkezi ve uzak oturumları destekleyen bir yöntemdir.

Merkezileştirilmiş ve uzak loglama, tüm önemli kayıtların güvenle tutulmasını sağlar. Docker şu anda çeşitli loglama sürücülerini destekliyor. Ortamınıza en uygun olanı kullanın. Docker info'yu çalıştırın ve Logging Driverproperty öğesinin uygun şekilde ayarlandığından emin olun.

```
$ docker info --format '{{.LoggingDriver}}'
```

Adım 1: Dokümanları takip ederek istenen log sürücüsünü kurun.
Adım 2: Docker daemonunu o loglama sürücüsüyle birlikte başlatın.

Örneğin,

```
$ dockerd --log-driver = syslog --log-opt syslog-address = tcp: //192.xxx.xxx.xxx
```

1. https://docs.docker.com/engine/admin/logging/overview/

Live Restore Seçeneğinin Açılması

--live-restore, docker'daki daemonsuz konteynerlere tam destek sağlamaktadır. Docker'ın kapatma veya geri yükleme sırasında konteynerleri durdurmadığından ve yeniden başlatıldığında konteynerlere yeniden bağlanmasını sağlamaktadır.

Docker daemonunda yer alan --live-restore parametresi, docker daemon mevcut olmadığında konteyner uygulamasının kesintiye uğramamasını sağlar. Bu aynı zamanda update veya patch sırasında downtime olmadan yapılabilmesini sağlar.

```
$docker info --format '{{ .LiveRestoreEnabled }}'
```

Aktifleştirmek için:

```
$ dockerd --live-restore
```

1. https://docs.docker.com/engine/admin/live-restore/

Userland Proxy'nin Kapatılması

Docker daemon, bir port her açıldığında (expose) port yönlendirme için bir userland proxy başlatır. NAT'ın mevcut olduğu yerlerde, bu servis genellikle gereksinimler için gereksizdir ve devre dışı bırakılabilir.

```
$ ps -ef | grep dockerd
```

Komutu ile --userland-proxy parametresini kontrol edebilirsiniz.

Kapatmak için aşağıdaki şekilde başlatın:

```
$ dockerd --userland-proxy=false
```

1. http://windsock.io/the-docker-proxy/
2. https://github.com/docker/docker/issues/14856
3. https://github.com/docker/docker/issues/22741
4. https://docs.docker.com/engine/userguide/networking/default_network/binding/

Daemon genelinde özel Seccomp Profilinin Uygulanması

Gerekirse, özel seccomp profilinizi daemon geneli düzeyinde uygulayabilir ve Docker'ın varsayılan seccomp profili geçersiz kılınabilmektedir. Çok sayıda sistem çağrısı, her bir userland processine tutunur ve çoğu işlemin tüm ömrü boyunca kullanılmaz. Uygulamaların çoğu, tüm sistem çağrılarına ihtiyaç duymaz ve bu nedenle, daha az sayıda mevcut sistem çağrısına sahip olmakla faydalanır. Azaltılmış sistem çağrıları seti, uygulamanın kullandığını toplam çekirdek yüzeyini azaltır ve böylece uygulama güvenliğini geliştirir.

Docker'ın varsayılan seccomp profili yerine kendi özel seccomp profilinizi uygulayabilirsiniz. Alternatif olarak, Docker'in varsayılan profili ortamınız için uygunsa, bu öneriyi yok saymayı seçebilirsiniz. Aşağıdaki komutu çalıştırın ve Security Options bölümünde listelenen seccomp profilini gözden geçirin. Varsayılan ise, yani Docker'ın varsayılan seccomp profile uygulanır.

```
$ docker info --format '{{.SecurityOptions}}'
```

Varsayılan olarak, Docker'ın varsayılan seccomp profili uygulanır. Bu, ortamınız için iyi ise, hiçbir işlem gerekli değildir. Alternatif olarak, kendi seccomp profilinizi seçmeyi seçerseniz, daemon başlangıcında --seccomp-profile parametresini kullanın veya daemon runtime parametreleri dosyasına koyun.

```
$ dockerd --seccomp-profile </ path / to / seccomp / profile>
```

Yanlış yapılandırılmış bir seccomp profili konteyner ortamınızı bozabilir. Docker varsayılan engellenen çağrılar dikkatlice incelenmiştir. Bunlar, konteyner ortamlarındaki bazı kritik güvenlik açıklarını / sorunları ele alır. Yani, varsayılanları geçersiz kılarken çok dikkatli olmalısınız.

Konteynerlerin Yeni Ayrıcalıklar Edinmesinin Kısıtlanması

Bir process "no_new_priv" bitini kernel'da ayarlayabilir. "No_new_priv" biti, processlerin veya child processlerrin, "suid" veya "sgid" bitleri aracılığıyla herhangi bir ek ayrıcalık kazanmamasını engeller. Böylelikle birçok tehlikeli operasyon çok daha az tehlikeli hale gelmektedir çünkü binary dosyaları bozmasını engleler. Bunu daemon seviyesinde ayarlayarak tüm yeni konteynerlerin varsayılan olarak yeni ayrıcalıkların elde edilmesini kısıtlanmış olmaktadır.

Sağlamak için:

```
$ dockerd --no-new-privileges
```

1. https://github.com/moby/moby/pull/29984
2. https://github.com/moby/moby/pull/20727

Docker Daemon Sıkılaştırmaları

Bu bölüm Docker ile ilgili dosya ve dizinlerin izinlerini ve sahipliğini kapsamaktadır. Hassas parametreler içerebilen dosya ve dizinlerin saklanması, Docker daemonunun doğru ve güvenli çalışması için önemlidir.

“Docker.Service” Dosyası Sahipliğinin Root:Root Olması

docker.service dosyası Docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Docker.service dosyasını root:root olarak düzenlemek için:

```
$ chown root:root /usr/lib/systemd/system/docker.service
```

“Docker.service” Dosya İzinlerinin 644 Veya Daha Kısıtlayıcı Olarak Ayarlanması

docker.service dosyası Docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için root dışındaki herhangi bir kullanıcı tarafından yazılabilir olmamalıdır.

Düzenlemek için:

```
$ chmod 644 /usr/lib/systemd/system/docker.service
```

“Docker.Socket” Dosyası Sahipliğinin Root:Root Olması

docker.socket dosyası, Docker API'sinin davranışlarını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root /usr/lib/systemd/system/docker.socket
```

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon- socket-option
2. https://github.com/docker/docker-ce/blob/master/components/packaging/deb/systemd/docker.socket

"Docker.Socket" Dosya İzinlerinin 644 Veya Daha Kısıtlayıcı Olarak Ayarlanması

docker.socket dosyası Docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için root dışındaki herhangi bir kullanıcı tarafından yazılabilir olmamalıdır.

Düzenlemek için:

```
$ chmod 644 /usr/lib/systemd/system/docker.socket
```

1. https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker- to-another-hostport-or-a-unix-socket
2. https://github.com/YungSang/fedora-atomic- packer/blob/master/oem/docker.socket
3. http://daviddaeschler.com/2014/12/14/centos-7rhel-7-and-docker-containers- on-boot/

"/etc/docker" Dizini Sahipliğinin Root:Root Olması

"/etc/docker" dizini çeşitli hassas dosyalara ek olarak sertifikalar ve anahtarlar içerir. Bu nedenle, dizinin bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root /etc/docker
```

https://docs.docker.com/engine/security/https/

"/etc/docker" Dizin İzinlerinin 755 Veya Daha Kısıtlayıcı Olarak Ayarlanması

/etc/docker dizini çeşitli hassas dosyalara ek olarak sertifikalar ve anahtarlar içerir. Bu nedenle, yalnızca dizinin bütünlüğünü korumak için kök tarafından yazılabilir olmalıdır.

Düzenlemek için:

```
$ chmod 755 /etc/docker
```

1. https://docs.docker.com/engine/security/https/

Registry Sertifikasının Sahipliğinin Root:Root Olması

/etc/docker/certs.d/ <kayıt defteri-adı> dizini Docker kayıt defteri (registry) sertifikalarını içerir. Sertifikaların bütünlüğünü korumak için bu sertifika dosyaları sahiplenmeli ve gruba ait olmalıdır.

Düzenelemek için:

```
$ chown root:root /etc/docker/certs.d/<registry-name>/*
```

1. https://docs.docker.com/registry/insecure/

Registry Sertifikalarının Dosya İzinlerinin 444 Veya Daha Kısıtlayıcı Olması

Tüm kayıt defteri sertifikası dosyalarının (genellikle /etc/docker/certs.d/ <kayıt defteri-adı> dizininde bulunur) 444 veya daha fazla kısıtlayıcı izinlere sahip olduğunu doğrulayın.

Düzenlemek için:

```
$ chmod 444 /etc/docker/certs.d/<registry-name>/*
```

TLS CA Sertifikasının Dosya Sahipliğinin Root:Root Olması

TLS CA sertifika dosyası herhangi bir değişiklikten korunmalıdır. Verilen CA sertifikasına dayalı Docker sunucusunu doğrulamak için kullanılır. Bu nedenle, CA sertifikasının bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root <path to TLS CA certificate file>
```

TLS CA Sertifikasının Dosya İzinlerinin 444 Veya Daha Kısıtlayıcı Olması

TLS CA sertifika dosyası herhangi bir değişiklikten korunmalıdır. Verilen CA sertifikasına dayalı Docker sunucusunu doğrulamak için kullanılır. Bu nedenle, CA sertifikasının bütünlüğünü korumak için 444 izinlerine sahip olmalıdır.

Düzenlemek için:

```
$ chmod 444 <path to TLS CA certificate file>
```

Docker Server Sertifika Sahipliğinin Root:Root Olması

Docker sunucusu sertifika dosyası herhangi bir değişiklikten korunmalıdır. Verilen sunucu sertifikasına dayalı Docker sunucusunu doğrulamak için kullanılır. Bu nedenle, sertifikanın bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.
Düzenlemek için:

```
$ chown root:root <path to Docker server certificate file>
```

Docker Server Sertifikası Dosya İzinlerinin 444 Veya Daha Kısıtlayıcı Olması

Docker sunucusu sertifika dosyası herhangi bir değişiklikten korunmalıdır. Verilen sunucu sertifikasına dayalı Docker sunucusunu doğrulamak için kullanılır. Bu nedenle, sertifikanın bütünlüğünü korumak için 444 izinleri olmalıdır.

```
$ chmod 444 <path to Docker server certificate file>
```

Docker Server Sertifika Key Sahipliğinin Root:Root Olması

Docker sunucu sertifikası anahtar dosyası, herhangi bir kurcalama veya gereksiz okumalardan korunmalıdır. Docker sunucu sertifikası için özel anahtarı tutar. Bu nedenle, Docker sunucu sertifikasının bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root <path to Docker server certificate key file>
```

Docker Server Sertifika Key Dosya İzinlerinin 400 Veya Daha Kısıtlayıcı Olması

Docker sunucu sertifikası anahtar dosyası, herhangi bir kurcalama veya gereksiz okumalardan korunmalıdır. Docker sunucu sertifikası için özel anahtarı tutar. Bu nedenle, Docker sunucu sertifikasının bütünlüğünü korumak için 400 izinlerine sahip olmalıdır.

Düzenlemek için:

```
$ chmod 400 <path to Docker server certificate key file>
```

"Daemon.Json" Sahipliğinin Root:Root Olması

daemon.json dosyası, docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root /etc/docker/daemon.json
```

"Daemon.Json" Dosya İzinlerinin 644 Veya Daha Kısıtlayıcı Olması

daemon.json dosyası, docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için yalnızca kök tarafından yazılabilir olmalıdır.

Düzenlemek için:

```
$ chmod 644 /etc/docker/daemon.json
```

"/etc/default/docker" Sahipliğinin Root:Root Olması

/ etc / default / docker dosyası, docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için sahip olunması ve gruba ait olması gerekir.

Düzenlemek için:

```
$ chown root:root /etc/default/docker
```

“/etc/default/docker” Dosya İzinlerinin 644 Veya Daha Kısıtlayıcı Olması

/ etc / default / docker dosyası, docker daemon davranışını değiştirebilecek hassas parametreler içerir. Bu nedenle, dosyanın bütünlüğünü korumak için yalnızca kök tarafından yazılabilir olmalıdır.

Düzenlemek için:

```
$ chmod 644 /etc/default/docker
```

Konteyner İmajları Ve İmaj Build Sıkılaştırmaları

Konteyner imajları ve imaj build dosyaları, konteynerlerin nasıl davranacağının temellerini oluşturur. Uygun imajlar ve uygun build dosyalarını kullandığınızdan emin olmak konteyner altyapınızı oluşturmak için çok önemli olmaktadır. Aşağıda, konteyner imajları için izlemeniz gereken önerilerden bazıları ve konteynerize edilmiş altyapınızın güvenli olduğundan emin olmak için bazı öneriler bulunmaktadır.

Dockerfile içerisinde Root Dışında Bir Kullanıcı Tanımlamak

Mümkünse, konteyneri root olmayan bir kullanıcı olarak çalıştırmak iyi bir uygulamadır. User namespace eşleştirmesi artık kullanılabilir olsa da, bir kullanıcı konteyner imajında zaten tanımlanmışsa, konteyner varsayılan olarak bu kullanıcı olarak çalıştırılır ve belirli namespaceleri yeniden eşlemesi gerekmez.

Örneğin, Dockerfile içerisine aşağıdaki gibi bir satır eklemek konteynerinizi oluştururken bir kullanıcı eklemenizi sağlar.

```
RUN useradd -d /home/username -m -s /bin/bash username USER username
```

Varsayılan olarak, kapsayıcılar root yetkileri ile çalıştırılır ve kullanıcı olarak root kullanır.

Güvenilir Konteyner Base İmajlarının Kullanılması

Resmi depolar, Docker topluluğu veya satıcı tarafından düzenlenmiş ve optimize edilmiş imajlardır. Potansiyel olarak güvenli olmayan diğer kamu depoları da kullanılabilir. Docker'dan ve üçüncü şahıslardan kuruluşunuzun verileri için nasıl kullanılacağına dair konteyner imajları indirilirken dikkatli olunmalıdır.

Konteynerlere Gereksiz Paketlerin Yüklenmemesi

Gereksiz yazılımlara sahip şişkin konteynerler ve konteynerlerin saldırı yüzeyini artırabilmektedir. Bu aynı zamanda konteyner imajlarının minimal olmasını da engellemektedir. Bu nedenle, konteynerin amacı için gerçekten gerekli olandan başka hiçbir şey yüklemeyin.

Konteynerlerin Güvenlik Taramalarından Geçirilmesi Ve Patchleme

İmajlar herhangi bir güvenlik açığı için "sık sık" olarak taranmalıdır. İmajlar yamaları içerecek şekilde yeniden oluşturulmalı ve yeniden başlatılmalıdır.

"HEALTHCHECK" Parametresinin Aktifleştirilmesi

Çalışan konteyner kontrollerini gerçekleştirmek için docker konteyner imajlarına HEALTHCHECK talimatı ekleyin.

Kontrol için:

```
$ docker inspect --format='{{ .Config.Healthcheck }}' <IMAGE>
```

Setuid and Setgid İzinlerinin İmajlardan Silinmesi

İmajlarda setuid ve setgid izinlerinin kaldırılması, konteynerde yetksi yükseltme saldırılarını önleyecektir. Setuid ve setgid izinleri yetki yükseltmek için kullanılabilir. Bu izinler zaman zaman gerekli olsa da, potansiyel olarak yetki yükseltme saldırılarında kullanılabilmektedir. Bu nedenle, bu izinler onlara ihtiyaç duyulmadığı sürece kullanılmamalıdır.

Setuid ve SetGid izinlerini bulunan çalıştırılabilir dosyaları bulmak için:

```
$ docker run <Image_ID> find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null
```

Dockerfile'ın sonuna doğru aşağıdaki komutu ekleyerek, oluşturma sırasında bu izinleri kaldırabilirsiniz:

```
RUN find / -perm +6000 -type f -exec chmod a-s {} \; || true
```

1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf
2. http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf
3. http://man7.org/linux/man-pages/man2/setuid.2.html
4. http://man7.org/linux/man-pages/man2/setgid.2.html

Dockerfile İçerisinde “ADD” yerine “COPY” Kullanımı

COPY komutu, dosyaları yerel ana bilgisayardan konteyner dosya sistemine kopyalar. "ADD" komutu, potansiyel olarak uzak URL'lerden dosya alabilir ve paket açma gibi işlemleri gerçekleştirebilir. Bu nedenle, ADD komutu, URL'lerden zararlı yazılımları eklemek gibi riskleri ortaya çıkarır.

Önlem:

ADD yerine COPY kullanılması gerekmektedir.

1. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best- practices/#add-or-copy

Doğrulanmış Paketlerin Kullanımı

İmajları yüklemeden önce paketlerin doğrulanması gerekmektedir. Paketleri doğrulamak, güvenli bir konteyner imajı oluşturmak için gereklidir. Bozuk paketler potansiyel olarak zararlı olabilir veya kötüye kullanılabilecek bazı bilinen güvenlik açıkları bulundurabilir.

Konteyner Çalışma Zamanı Sıkılaştırmaları

AppArmor'un Aktifleştirilmesi

AppArmor etkili ve kullanımı kolay bir Linux uygulama güvenlik sistemidir. Debian ve Ubuntu gibi varsayılan olarak birkaç Linux dağıtımında kullanılabilir. AppArmor, aynı zamanda AppArmor profili olarak da bilinen güvenlik politikasını uygulayarak uygulamaları çeşitli tehditlerden korur. Konteynerler için kendi AppArmor profilinizi oluşturabilir veya Docker'ın varsayılan AppArmor profilini kullanabilirsiniz.

Kontrol için:

```
$ docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: AppArmorProfile={{
.AppArmorProfile }}'
```

AppArmor Linux işletim sisteminiz için uygunsa, onu kullanın. Aşağıdaki adımları uygulayın:

1. AppArmor'un kurulu olup olmadığını kontrol edin. Değilse, kurun.
2. Docker konteynerleri için bir AppArmor profili oluşturun veya bulun.
3. Bu profili zorlama (encorfing) moduna geçirin.
4. Docker kotneynerinizi özelleştirilmiş AppArmor profilini kullanarak başlatın. Örneğin,

```
$ docker run --interactive --tty --security-opt="apparmor:PROFILENAME" centos /bin/bash
```

1. https://docs.docker.com/engine/security/apparmor/
2. https://docs.docker.com/engine/reference/run/#security-configuration
3. https://docs.docker.com/engine/security/security/#other-kernel-security-features

SELinux'un Aktifleştirilmesi

SELinux etkili ve kullanımı kolay bir Linux uygulama güvenlik sistemidir. Red Hat ve Fedora gibi varsayılan olarak birkaç Linux dağıtımında mevcuttur. SELinux, varsayılan İsteğe Bağlı Erişim Denetimi (DAC) modelini büyük ölçüde artıran bir Zorunlu Erişim Denetimi (MAC) sistemi sağlar. Bu nedenle, eğer varsa, Linux sunucunuzda SELinux'u etkinleştirerek fazladan bir güvenlik katmanı ekleyebilirsiniz.

Kontrol:

```
$ docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{
.HostConfig.SecurityOpt }}'
```

SELinux Linux işletim sisteminiz için uygunsa, kullanın. Aşağıdaki adımları uygulamanız gerekebilir:

1. SELinux Durumunu ayarlayın.
2. SELinux Politikasını ayarlayın.
3. Docker konteynerler için bir SELinux şablonu oluşturun veya bulun.
4. SELinux etkin olarak Docker daemon modunu başlatın. Örneğin,

```
$ docker daemon --selinux-enabled
```

Konteyner için:

```
$docker run --interactive --tty --security-opt label=level:TopSecret centos /bin/bash
```

1. https://docs.docker.com/engine/security/security/#other-kernel-security-features
2. https://docs.docker.com/engine/reference/run/#security-configuration
3. http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/
4. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy

Privileged (Ayrıcalıklı) Konteynerlerin Kullanılmaması

“--privileged” parametresini kullanmak, tüm Linux Kernel kabiliyetlerini konteynere verir ve böylece "--cap-add" ve "--cap-drop" parametreleri geçersiz olur. Kullanılmadığından emin olun.

“Privileged” parametresi tüm yetenekleri konteynere verir ve tüm sınırlamaları kaldırır. Başka bir deyişle, konteyner host(ana) makinenin yapabileceği neredeyse her şeyi yapabilir. Bu parametre, Docker'da Docker'ı çalıştırmak gibi özel kullanım durumlarına izin vermek için var.

Kontrol için:

```
$ docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Privileged={{
.HostConfig.Privileged }}'
```

Önlem: --privileged parametresi ile konteynerleri çalıştırmayın, örneğin:

```
$ docker run --interactive --tty --privileged centos /bin/bash
```

Ana (host) Sunucunun Sistem Dizinlerinin Mount Edilmemesi

Aşağıdaki gibi hassas ana sunucu sistem dizinlerinin özellikle okuma-yazma (read-write) modunda konteyner birimlere olarak mount edilmesine izin verilmemelidir.

```
/
/boot
/dev
/etc
/lib
/proc
/sys
/usr
```

Hassas dizinler okuma-yazma modunda mount edilirse, bu hassas dizinlerdeki dosyalarda değişiklik yapmak mümkün olabilir. Değişiklikler, Docker ana sunucusunu tehlikeye sokabilir ve gereksiz değişikliklere neden olabilir.

Kontrol için:

```
$ docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}
```

Konteyner içerisinde SSH Servisinin Kapatılması

SSH konteynerde çalıştırılmamalıdır. Docker ana bilgisayarına SSH yapmalı ve uzak bir ana bilgisayardan bir konteynere girmek için nsenter aracını kullanabilirsiniz.

1. http://blog.docker.com/2014/06/why-you-dont-need-to-run-sshd-in-docker/

Konteynerde Sadece Gerekli Portların Açık Olması

Bir konteyner, sadece Dockerfile dosyasında tanımlanan portlar ile çalıştırılabilir veya ilgili çalışma parametrelerinde belirtilen portları kullanabilir. Dockerfile çeşitli değişikliklere uğrayabilir ve açıkta bulunan bağlantı noktaları listesi konteyner içinde çalıştığınız uygulama ile ilgili olabilir veya olmayabilir. Gereksiz portların açılması konteynerin ve konteyner uygulamasının saldırı yüzeyini arttırır. Önerilen bir uygulama olarak, gereksiz portları açmayın.

Kontrol için:

```
$ docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{ .NetworkSettings.Ports }}'
```

1. https://docs.docker.com/engine/userguide/networking/

Konteynerlerin HOST Networkünü Kullanmaması

--net=host parametresi ayarlandığında bir konteyner ağ modu devre dışı kalır. Bu seçenek Docker'a, konteynerin ağını kapatmasını söyler. Konteyner Docker ana bilgisayarında "dışarıda" çalışmasına sebep olur ve ana bilgisayarın ağ arabirimlerine tam erişime sahip olduğu anlamına gelir.

Bu potansiyel olarak tehlikelidir. Konteynerin diğer herhangi bir portu açmasına izin verir. Ayrıca konteynerin Docker ana bilgisayarındaki D-bus gibi ağ servislerine erişmesini de sağlar. Böylece, bir konteyner processi, Docker ana bilgisayarını kapatmak gibi beklenmedik şeyler yapabilir. Bu seçeneği kullanmamalısınız.

Kontrol için:

```
$ docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: NetworkMode={{
.HostConfig.NetworkMode }}'
```

Konteyner Trafiğinin Bir Network Arayüzüne Bind Edilmesi

Ana makinenizde birden çok ağ arabiriminiz varsa, konteyner, herhangi bir ağ arabirimindeki açılmış portlardan bağlantıları kabul edebilir. Bu istenmeyebilir ve güvenli olmayabilir. Bir arayüzün birden fazla arayüze eriştiği IDS,IPS, Firewall, Load balancer gibi hizmetler dışında gereksiz olmaktadır. Bu Hizmetler, gelen toplu trafiği taramak için bu arayüzler üzerinde çalıştırılır. Bu nedenle, herhangi bir arayüzden gelen tüm bağlantıları kabul etmemelisiniz. Sadece belirli bir arayüzden gelen bağlantılara izin vermelisiniz.

Kontrol ve Port Mapping için:

```
$ docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{ .NetworkSettings.Ports }}
```

Listeyi gözden geçirin ve açık konteyner portlarının belirli bir arayüze bağlı olduğundan emin olunuz - IP adresinin - 0.0.0.0 olmaması gerekmektedir.

Konteyneri spesifik bir ağ ve port için çalıştırmak için:

```
$ docker run --detach --publish 10.2.3.4:49153:80 nginx
```

Yukarıdaki örnekte, konteyner 80 portu 49153'teki host portuna bağlanır ve sadece 10.2.3.4 network arayüzden gelen bağlantıyı kabul eder.

1. https://docs.docker.com/engine/userguide/networking/

Docker0 Ağ Arayüzünün Kullanılmaması

Docker'ın varsayılan köprüsü docker0 kullanmayın. Konteyner ağ bağlantısı için docker'ın kullanıcı tanımlı ağlarını kullanın. Docker, köprü modunda oluşturulan sanal arabirimleri docker0 adlı ortak bir köprüye bağlar. Bu varsayılan ağ modeli, filtreleme uygulanmadığından ARP spoofing ve MAC flooding saldırılarına karşı savunmasızdır.

Kontrol için:

```
$ docker network ls --quiet | xargs xargs docker network inspect --format '{{ .Name }}: {{
.Options }}'
```

1. https://github.com/nyantec/narwhal
2. https://arxiv.org/pdf/1501.02967
3. https://docs.docker.com/engine/userguide/networking/

Docker’da Bulunan Geçmiş Zafiyetler

Docker'ın piyasaya sürülmesinden bu yana, işlevselliğini ve güvenliğini artırmak için bir dizi güncelleme ve değişiklik yapılmıştır Yine de, herhangi bir programcının bildiği gibi, kesinlikle güvenli bir platform yoktur ve Docker bir istisna değildir. Ayrıca, çoğu zaman, güvenlik, bir kullanıcının Docker ile nasıl etkileşim kurduğundan etkilenir ve bu da insani hatalarla ilgili birkaç sorunla karşılaşmaktadır.

2016 yılında Cloud Foundry (https://www.cloudfoundry.org/hope-versus-reality-containers-in-2016/) tarafından yürütülen çok sayıda gelişmiş ülkeden 700'ün üzerinde şirketin yer aldığı bir çalışmada şirketlerin yarısının konteyner teknolojsini kullandığını göstermektedir. Bunların arasında %64'ü ise konteyner teknolojisinin kullanımını yaygınlaştırmayı planlamaktadır. Konteyner kullanımındaki bu hızlı artışa bakıldığında, yeni güvenlik sorunları da sürekli olarak ortaya çıkmaktadır. Docker, günümüzde en çok kullanılan konteyner teknolojisidir ve diğer benzer platformlarda olduğu gibi, birçoğu protokole özgü olan güvenlik sorunlarına sahiptir.

Daha önce Docker’da bulunmuş olan zafiyetlerin listesi CVE-List üzerinden gözlemlenebilir ve bu güvenlik sorunları hakkında bilgi edinilebilir.

Geçmiş Zafiyetler Listesi:

- https://www.cvedetails.com/vulnerability-list/vendor_id-13534/product_id-28125/Docker-Docker.html

Kaynaklar:

1. https://rhelblog.redhat.com/2016/10/17/secure-your-containers-with-this-one-weird-trick/
2. https://www.cisecurity.org/benchmark/docker/
3. https://rhelblog.redhat.com/2016/10/17/secure-your-containers-with-this-one-weird-trick/
4. https://rhelblog.redhat.com/2017/01/13/selinux-mitigates-container-vulnerability/

BGA Bilgi Güvenliği A.Ş. Hakkında

BGA Bilgi Güvenliği A.Ş. 2008 yılından bu yana siber güvenlik alanında faaliyet göstermektedir. Ülkemizdeki bilgi güvenliği sektörüne profesyonel anlamda destek olmak amacı ile kurulan BGA Bilgi Güvenliği, stratejik siber güvenlik danışmanlığı ve güvenlik eğitimleri konularında kurumlara hizmet vermektedir.

Uluslararası geçerliliğe sahip sertifikalı 50 kişilik teknik ekibi ile, faaliyetlerini Ankara ve İstanbul ve USA'da sürdüren BGA Bilgi Güvenliği'nin ilgi alanlarını "*Sızma Testleri, Güvenlik Denetimi, SOME, SOC Danışmanlığı, Açık Kaynak Siber Güvenlik Çözümleri, Büyük Veri Güvenlik Analizi ve Yeni Nesil Güvenlik Çözümleri*" oluşturmaktadır.

Gerçekleştirdiği başarılı danışmanlık projeleri ve eğitimlerle sektörde saygın bir yer edinen BGA Bilgi Güvenliği, kurulduğu günden bugüne alanında lider finans, enerji, telekom ve kamu kuruluşlarına 1.000'den fazla eğitim ve danışmanlık projeleri gerçekleştirmiştir.

BGA Bilgi Güvenliği, kurulduğu 2008 yılından beri ülkemizde bilgi güvenliği konusundaki bilgi ve paylaşımların artması amacı ile güvenlik e-posta listeleri oluşturulması, seminerler, güvenlik etkinlikleri düzenlenmesi, üniversite öğrencilerine kariyer ve bilgi sağlamak için siber güvenlik kampları düzenlenmesi ve sosyal sorumluluk projeleri gibi birçok konuda gönüllü faaliyetlerde bulunmuştur.

BGA Bilgi Güvenliği AKADEMİSİ Hakkında

BGA Bilgi Güvenliği A.Ş.'nin eğitim ve sosyal sorumluluk markası olarak çalışan Bilgi Güvenliği AKADEMİSİ, siber güvenlik konusunda ticari, gönüllü eğitimlerin düzenlenmesi ve siber güvenlik farkındalığını arttırıcı gönüllü faaliyetleri yürütülmesinden sorumludur. Bilgi Güvenliği AKADEMİSİ markasıyla bugüne kadar "Siber Güvenlik Kampları", "Siber Güvenlik Staj Okulu", "Siber Güvenlik Ar-Ge Destek Bursu", "Ethical Hacking yarışmaları" ve "Siber Güvenlik Kütüphanesi" gibi birçok gönüllü faaliyetin destekleyici olmuştur.